Тема: Основні навички роботи в середовищі Cocos2d-x v. 3.x

Мета: Ознайомити слухачів з основними елементами середовища гри, навчитись створювати елементи гри та задавати головні початкові властивості та атрибути, такі як зовнішній вигляд, переміщення, послідовності.

Зміст:

1. Основні елементи та складові ігрового середовища проекту в Cocos2d-x v. 3.x.
2. Використання Sprites
3. Основні команди дії (Actions).
4. Створення послідовностей (Sequences та Spawn)
5. Створення батьківських вузлів
6. Завдання.

Основні елементи та складові ігрового середовища проекту в Cocos2d-x v. 3.x.

Cocos2d-x це крос-платформний ігровий фреймворк. Ігровий фреймворк надає загальну функціональність яка повинна бути у всіх ігор. Він включає в себе компоненти, які разом дозволяють прискорити розробку. Наприклад renderer, графіку, механізм виявлення зіткнень, фізику, звук, анімацію. Cocos2d-x надає простий API для розробки крос-платформних додатків (вірніше, дозволяє збирати додаток під різні платформи).

Cocos2d-x - це двигун із відкритим кодом під керуванням ліцензії MIT. Він може бути використаний для створення ігор, додатків та інших інтерактивних програм на основі крос-платформних графічних інтерфейсів.

Cocos2d-x широко використовується людьми та ентузіастами, а також великими компаніями. На сьогоднішній день багато хто з ігор Cocos2d-x домінують у найпопулярніших графіках AppStore і Google Play, особливо в Китаї, Південній Кореї та Японії. Інженери з Chukong, Google, Microsoft, ARM, Intel та BlackBerry активно займаються спільнотою Cocos2d-x. Cocos2d-x використовував для розробки своїх ігор великі хлопці, такі як Zynga, Wooga, Glu, IGG, Big Fish Games, Fingersoft, Gamevil, GREE, DeNA, Konami, CJ E & M, NHN, LINE, Square Enix, Disney Mobile.

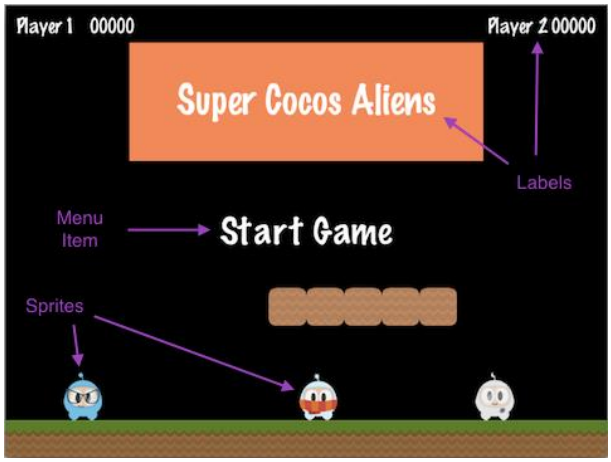
Cocos2d-x надає такі об'єкти як **Scene, Transition, Sprite, Menu, Sprite3D, Audio** і інші.

Головним компонентами є: Scene, Node, Sprite і Action.

Більшість ігор виглядає по різному, але кажна з них має стандартний набір елементів, які можуть відображатись по різному. Але в загальному випадку комп'ютерну гру можна звести до вигляду як зображено на мал.1:

мал. 1

Всі компоненти стандартної сцени зображені на мал. 2:



мал. 2

Система координат

Треба зауважити, що Cocos2d використовує декартову систему координат. Тобто точка (0, 0) знаходиться зліва внизу.

Director

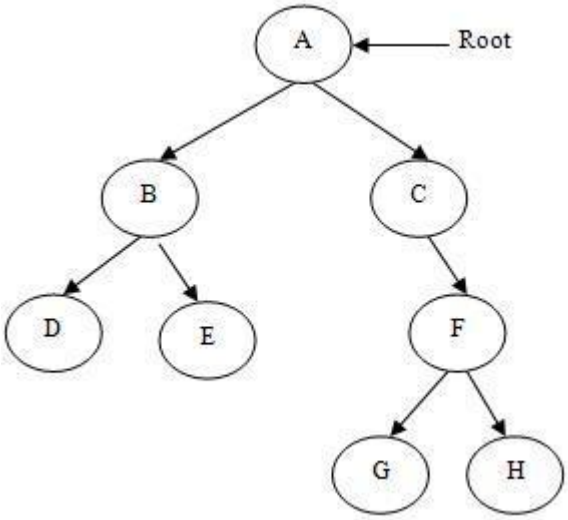
Director в Cocos2d-x це як режисер на знімальному майданчику. Він контролює всі об'єкти і каже їм, що вони повинні робити. Director управляє зміною сцен і ефектами переходу, він є доступний звідусіль.

Scene (сцена)

У грі швидше за все буде меню, кілька рівнів, і ще кілька екранів («Ви програли!», «Почати гру»!). Кожен такий екран - це сцена (Scene). Сцена відмальовується об'єктом Renderer. Renderer відповідає за побудову спрайтів і інших об'єктів сцени. Для кращого розуміння цього процесу, поговоримо про Scene Graph.

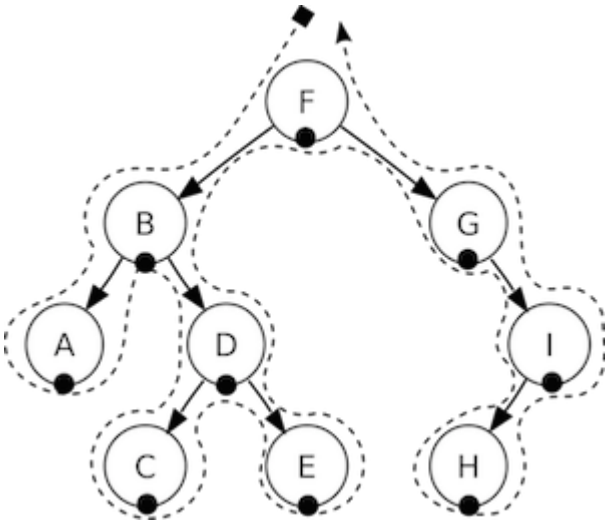
Scene Graph

Scene Graph це структура даних, яка використовується в Scene. Scene Graph містить вузли (Node). До речі кажучи, Scene Graph називається Scene Graph, але насправді це дерево.



мал. 3

Cocos2d-x виконує симетричний обхід дерева (при якому відвідується спочатку ліве піддерево, потім вузол, потім - праве піддерево). Це означає, що права частина дерева відобразиться останньою, і значить, буде видно «зверху» (тобто візуально інші - під нею).

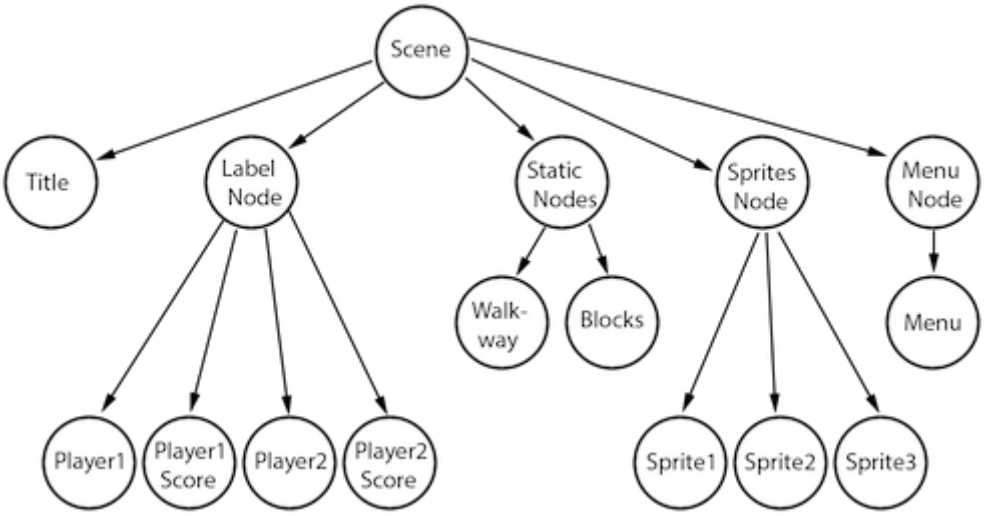


мал. 4

Проілюструвати це дуже легко, давайте просто поглянемо на сцену мал. 5:

мал. 5

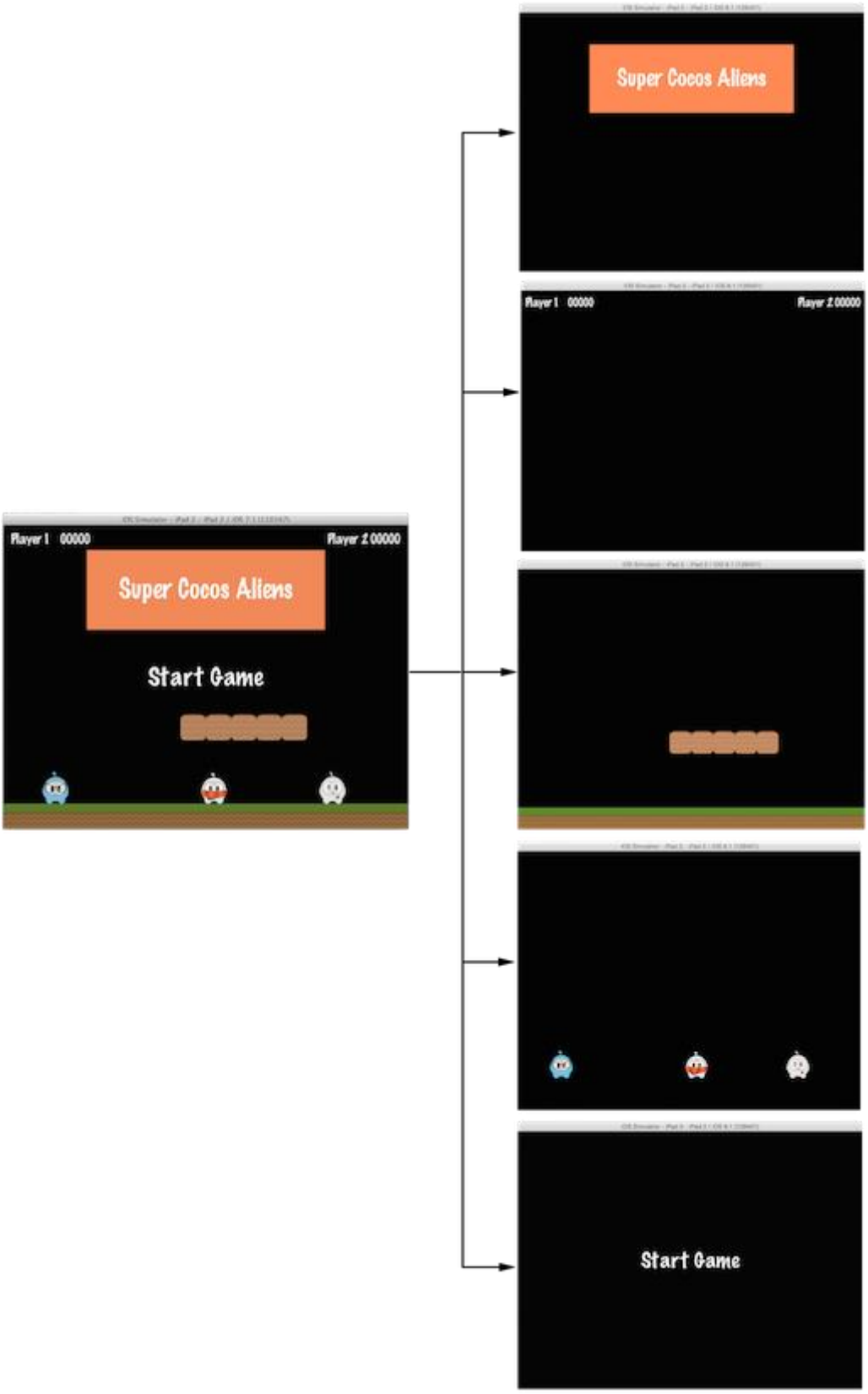
Спрощено дану схему (частина об'єктів не є листами, а гілками) можна представити у вигляді дерева:



мал. 6

Атрибут об'єктів пов'язаний з порядком це **z-order**. Ліва частина дерева має негативний z-order, в той час як права - позитивний. Можна тримати це в голові, щоб уникнути помилок. Плюс до цього, можна додавати елементи в будь-якому порядку з зазначеним z-order, тоді елементи будуть відсортовані автоматично.

Можна розглядати Scene як набір об'єктів (Node). Розіб'ємо сцену, показану вище, щоб подивитися scene graph (мал 7):



мал.7

Сцена зліва це складені разом вузли (Node) мають різний z-order, які визначають порядок їх відтворення.

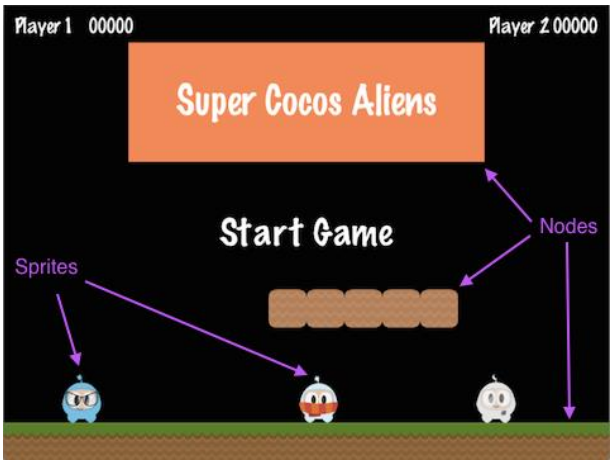
Додати елемент до сцени можна прописавши код:

```
// Додаємо елемент з z-order -2,
// він буде в лівій частині дерева
scene->addChild(title_node, -2);

// Якщо не вказати z-order явно,
//використовується значення за замовчуванням – 0

scene->addChild(label_node);

// Додаємо елемент з z-order 1,
// він буде в правій частині дерева
scene->addChild(sprite_node, 1);
```

Використання Sprites

Всі ігри використовують спрайти. Це ті штуки, які рухаються по екрану. Ви можете управляти ними. Головний герой гри, скоріше за все, є спрайтом. Важливо відзначити: не кожен графічний елемент гри це спрайт. Якщо елемент не переміщається по екрану, то це просто вузол (Node).

Погляньмо ще раз на сцену з гри на якій зображено як статичні об'єкти так і динамічні. Основними атрибутами, очевидно є спрайти (мал. 8):



мал. 8

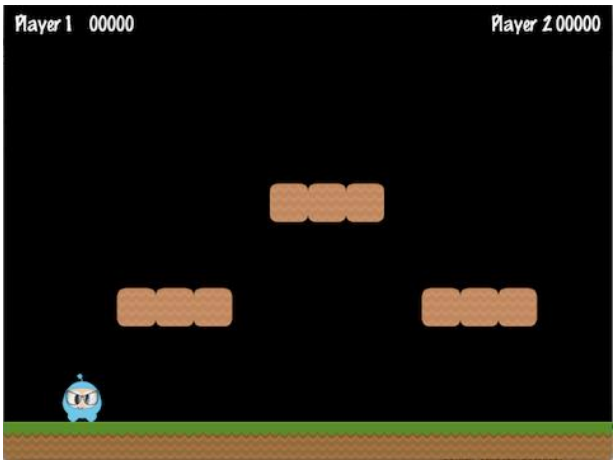
Основні команди для роботи зі спрайтами:

Таблиця 1. Основні команди для спрайту

`auto mySprite = Sprite::create("1.png");`	Створити спрайт, з зображенням із файла 1.png в папцій Resources
`scene->addChild(mySprite);`	Добавлення спрайта на сцену
`mySprite->setScale(2.0);`	Збільшення розмірів спрайта (>1 збільшення, <1 зменшення
`mySprite->setPosition(Vec2(200, 200));`	Встановлення спрайту в визначені координати
`mySprite->setRotation(40);`	Поворот спрайта, за годинниковою стрілкою

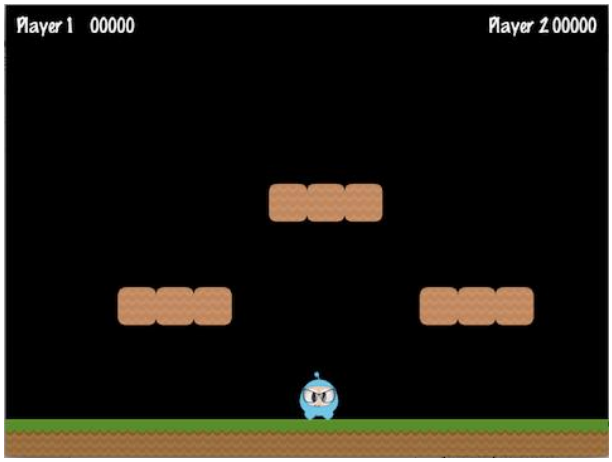
mySprite->setAnchorPoint(Vec2(0, 0));	Вказує яку частину спрайту привязувати до положення, при використанні команди setPosition (значення, 0, 0.5, 1) 0 0 – нижній лівий кут, … 1 1 – верхній правий кут
mySprite->runAction('name')	Задає дію, яка прописана в об'єкті в дужках. (в дужках може бути об'єкт класу MoveBy, MoveTo)

Спробуємо проілюструвати основні команди для роботи зі спрайтами
Стартове вікно проекту:



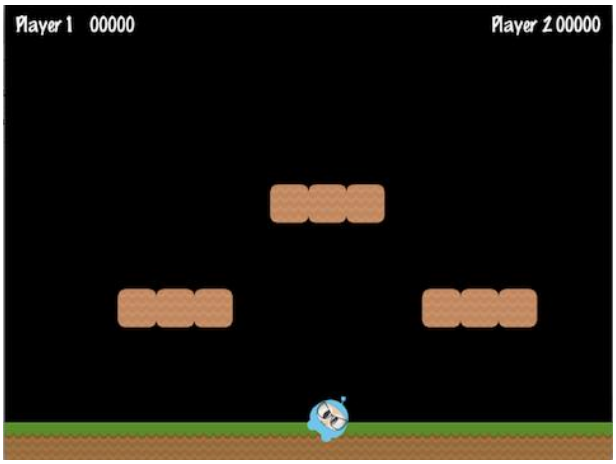
мал. 9

Встановлення спрайту (мал. 10): **mySprite->setPosition(Vec2(500, 0));**



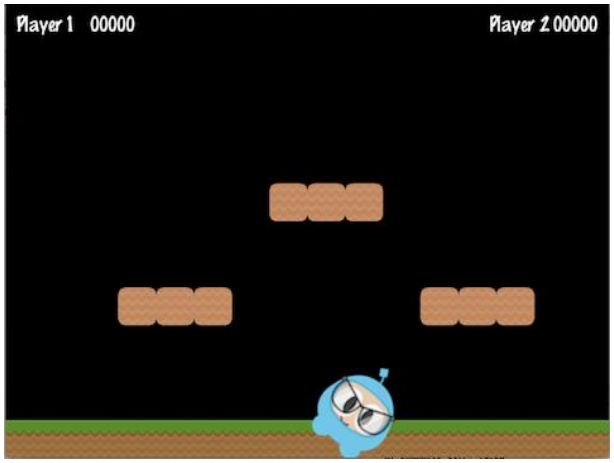
мал. 10

Поворот спрайту (мал.11): **mySprite->setRotation(40);**:



мал. 11

Масштабування розмірів спрайту (мал. 12): **mySprite->setScale(2.0);**:

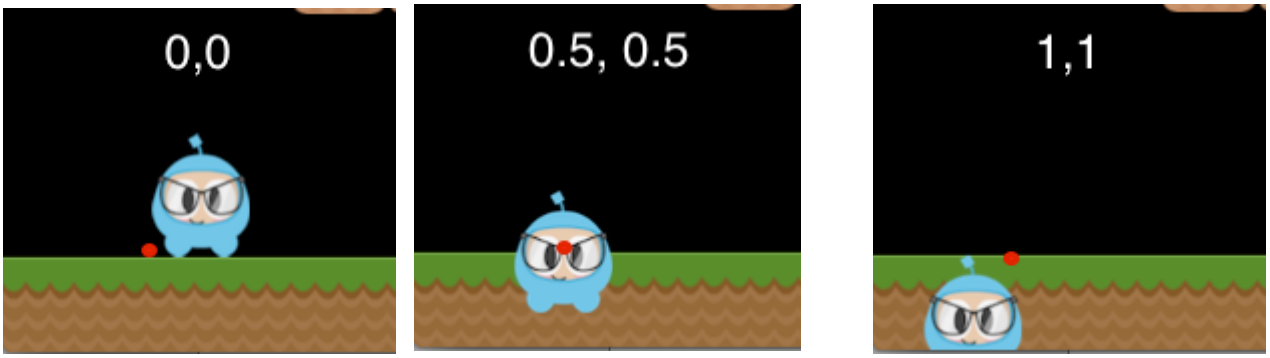


мал. 12

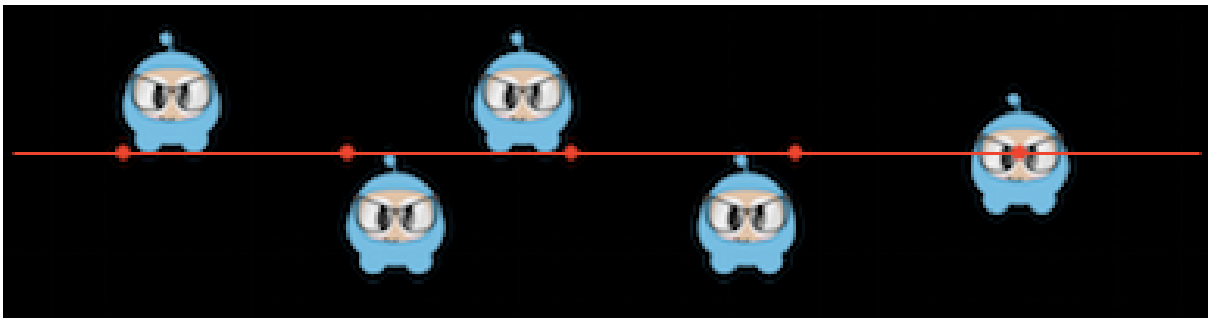
Всі об'єкти вузли (Sprite є підкласом вузла) мають значення для точки прив'язки. Можете задати точку прив'язки вузла, тобто вказати яку частину спрайту використовувати як базову координату при встановленні її положення.

Для цього існує команда (мал. 13-14) **mySprite->setAnchorPoint(Vec2(0, 0));**

Результати дії команди з різними значеннями аргументів можна переглянути на малюнках:



мал. 13



мал. 14

Червона точка ілюструє, де знаходиться опорна точка!

Як видно, точка прив'язки дуже корисна при позиціонуванні спрайтів. Опорну точку можна налаштовувати динамічно, щоб імітувати ефекти у проекті.

Основні команди дії (Actions).

Створення сцени з додавання спрайтів на екран це тільки частина завдання. Важливим елементом гри є рух об'єктів. Для цього й існують екшени (Actions). Рух, повороти, обертання - це все екшени.

Основні команди для переміщення спрайтів по сцені наведені в таблиці 2.

Таблиця 2. Команди переміщення спрайту

`auto moveBy = MoveBy::create(t, Vec2(x, y));`	Створення об'єкта класу MoveBy , який означає зміну положення координат на вказані значення **x, y**, за час **t сек.** *Для використання одного і тогож об'єкту до різних спрайтів потрібно використовувати метод clone()* ***напр.*** **moveBy->clone()**.
`auto moveTo = MoveTo::create(t, Vec2(x, y));`	Створення об'єкта класу MoveTo , який означає зміну положення до вказаних значення **x, y** , за час **t сек** *Для використання одного і тогож об'єкту до різних спрайтів потрібно використовувати метод clone()* ***напр.*** **moveBy->clone()**.

Створення послідовностей (Sequences)

Вищеописані команди дозволяють виконати поодинокі рухи спрайтів, в більшості випадків, рух це складний елемент та передбачає сукупність елементарних переміщень. Організувати сукупності переміщень спрайтів по сцені можна за допомогою послідовностей.

Послідовність - це декілька дій над об'єктами, які запускаються у вказаному порядку.

Як приклад, розглянемо рух об'єкту за наступною схемою (мал. 15):

мал .15

Створення послідовності, має наступний код:

```
auto mySprite = Node::create();
// рух до вказаних координат за 2 сек.
auto moveTo1 = MoveTo::create(2, Vec2(50, 10));
// рух до координат, які віддалені від поточних на 100 по x та 10 по y за 2 сек
auto moveBy1 = MoveBy::create(2, Vec2(100, 10));
// рух до вказаних координат за 2 сек.
auto moveTo2 = MoveTo::create(2, Vec2(150, 10));
// створення паузи
auto delay = DelayTime::create(1);
// формування заданої послідовності дій
mySprite->runAction(Sequence::create(moveTo1, delay, moveBy1, delay.clone(),
    moveTo2, nullptr));
```

Слід звернути увагу на повторне використання об'єкту delay Створені об'єкти руху можна використовувати тільки один раз, це стосується і moveTo, MoveBy. Для повторного використання об'єкту потрібно використовувати метод clone(), як це показано в прикладі delay.clone()

У наведеному прикладі всі вказані дії виконуються постлідовно, проте можуть виникнути випадки, коли дії потрібно виконати одночасно. В такому випадку використовується не **Sequence**, а **Spawn**.

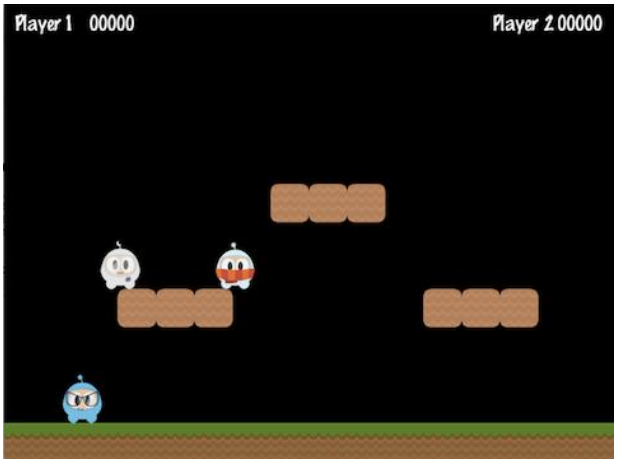
Деякі з них можуть бути довшими за інших, тому вони не закінчаться одночасно.

```
auto myNode = Node::create();

auto moveTo1 = MoveTo::create(2, Vec2(50, 10));
auto moveBy1 = MoveBy::create(2, Vec2(100, 10));
auto moveTo2 = MoveTo::create(2, Vec2(150, 10));

myNode->runAction(Spawn::create(moveTo1, moveBy1, moveTo2, nullptr));
```

Створення батьківських вузлів

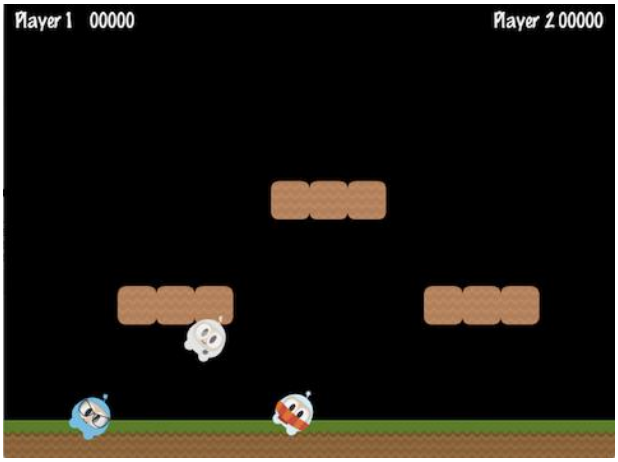
Cocos2d-x використовує відносини між батьками та дитиною (базовими та наслідуваними вузлами). Це означає, що властивості та зміни батьківського

вузла застосовуються до її дітей. Розглянемо єдиний спрайт, а потім спрайт, який має дітей:



мал. 16

При застосуванні обертання до батьківського вузла, такіж дії поширяться на всі Child вузли.



мал. 17

```
auto myNode = Node::create();
// поворот
myNode->setRotation(50);
```

Аналогічні дії можна зробити із зміною маштабу спрайтів:

мал. 18

```
auto myNode = Node::create();
myNode->setScale(2.0);
```

Не всі зміни батьків передаються дітям. Зміна батьківської точки прив'язки впливає лише на операції перетворення (маштабування, положення, поворот, перекочування тощо (scale, position, rotate, skew, etc...)) і не впливає на позиціонування дітей. Фактично діти завжди додадуться до нижнього лівого (0,0) кута їхнього батьківського вузла.

Для того щоб додати до батьківського вузла Child вузол, слід виконати метод AddChild:

```
auto myNode = Node::create();
auto mySprite = Sprite::create("2.png");
myNode->addChild(mySprite);
```

Для прив`язки Child вузла до іншого батьківського елемента виконується наступні команди:

```
//відв'язка від попереднього батьківського елемента
sprite->retain();
sprite->removeFromParent();
// привязка до нового елемента
newParent->addChild(sprite);
sprite->release();
```

Завдання

Створити проект з фоновим малюнком у вигляді схеми доріг. Запрограмувати спрайт, який би рухався виключно по дорозі. Реалізувати можливість руху з різними швидкостями.

Список літературних джерел по темі:

1. Siddharth Shekar Learning Cocos2d-x Game Development // 2014 Packt Publishing, p.266, ISBN 978-1-78398-826-6
2. Raydelto Hernandez Building Android Games with Cocos2d-x // 2015 Packt Publishing, p. 160, ISBN 978-1-78528-383-3
3. Roger Engelbert Cocos2d-x by Example Beginner's Guide Second Edition // 2015 Packt Publishing, p. 270, ISBN 978-1-78528-885-2
4. Akihiro Matsuura Cocos2d-x Cookbook // 2015 Packt Publishing, p. 255, ISBN 978-1-78328-475-7
5. Karan Sequeira Cocos2d-x Game Development Blueprints // 2015 Packt Publishing, p. 392, ISBN 978-1-78398-526-5
6. Frahaan Hussain, Arutosh Gurung, Gareth Jones Cocos2d-x Game Development Essentials // 2014 Packt Publishing, p. 392, ISBN 978-1-78398-786-3